Hindawi Publishing Corporation
Mobile Information Systems
Volume 2015, Article ID 241732, 9 pages
http://dx.doi.org/10.1155/2015/241732

Hindawi

Research Article

An Efficient SDN Load Balancing Scheme Based on Variance Analysis for Massive Mobile Users

Hong Zhong,[1] Qunfeng Lin,[1] Jie Cui,[1] Runhua Shi,[1] and Lu Liu[2]

[1]*School of Computer Science and Technology, Anhui University, Hefei 230039, China*
[2]*Department of Computing and Mathematics, University of Derby, Kedleston Road, Derby DE22 1GB, UK*

Correspondence should be addressed to Jie Cui; cuijie@mail.ustc.edu.cn

Received 10 October 2015; Accepted 6 December 2015

Academic Editor: Mianxiong Dong

In a traditional network, server load balancing is used to satisfy the demand for high data volumes. The technique requires large capital investment while offering poor scalability and flexibility, which difficultly supports highly dynamic workload demands from massive mobile users. To solve these problems, this paper analyses the principle of software-defined networking (SDN) and presents a new probabilistic method of load balancing based on variance analysis. The method can be used to dynamically manage traffic flows for supporting massive mobile users in SDN networks. The paper proposes a solution using the OpenFlow virtual switching technology instead of the traditional hardware switching technology. A SDN controller monitors data traffic of each port by means of variance analysis and provides a probability-based selection algorithm to redirect traffic dynamically with the OpenFlow technology. Compared with the existing load balancing methods which were designed to support traditional networks, this solution has lower cost, higher reliability, and greater scalability which satisfy the needs of mobile users.

1. Introduction

In considering network overhead, techniques for load balancing are of significant importance. Load balancing directly impacts application and service availability for mobile users [1]. Load balancing aims to optimize the utilization of the resource by maximizing the throughput, minimizing the response time, and avoiding overloading of any single resource. To alleviate heavy-traffic network flux and reduce the risk that a single server will become the main overhead contributor, many data centers adopt dedicated hardware methods to enable load balancing in order to support a large number of users [2, 3]. However, the hardware systems are usually expensive to procure, can be technically challenging to be deployed, and may require human intervention to work consistently.

Software-defined networking (SDN) as a type of computer networking provides a simple, convenient, maneuverable network flow control method with a minimal investment so as to reduce cost and increase benefit for massive mobile users. It controls data transport by means of software implementation of switches. When a data flow arrives at a switch, a flow table lookup has to be carried out. Flow tables ([Header : Counters : Actions]) are widely used in SDN. For each network flow, the headers and counters will be updated if flow changes are required or actions are imposed. By recording the header information into a database, an OpenFlow switch can process the data flow according to the header records. Based on the SDN model with a centralized controller, an OpenFlow switch is designed for different rules to control the network traffic using the header records. The flow control system will theoretically make it possible to define an algorithm to balance the network load.

This paper aims to present a new probabilistic method of load balancing based on variance analysis in SDN networks for supporting dynamic demand from massive mobile users. The SDN controller can monitor data traffic of each server port and manage all inbound and outbound traffic from server clusters. By deploying dynamically extensible load balancing strategy, an efficient model is proposed to reduce the packet latency in traditional communication networks and guarantee the reliability for massive mobile users, continuity, and timeliness of their business. In comparison to existing load balancing methods, the proposed method is able to solve

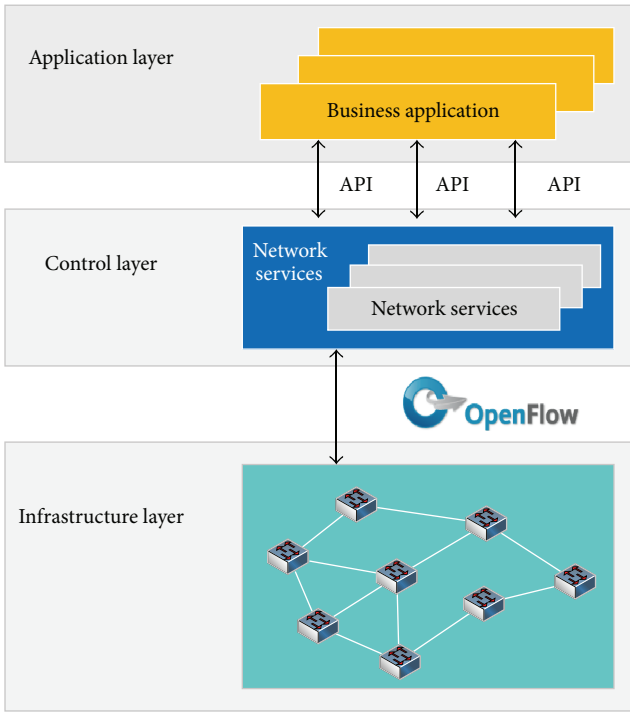

Figure 1: The SDN architecture.

the observed deficiencies of traditional methods such as high cost, low reliability, and poor extensibility.

2. Related Work

SDN (software-defined networking) is a technology in the field of computer networking which is presently generating significant interest. It originated from a project that began at UC Berkeley and Stanford University around 2008 [4]. SDN is currently seen as one of the emerging approaches to computer networking that allows network researchers to manage network services through abstraction of lower level functionality [5, 6]. This is achieved by decoupling the network control that makes decisions about where traffic is sent (the control plane) from the forwarding systems that forward traffic to the selected destination (the data plane). The network becomes directly programmable and allows the infrastructure to be abstracted for applications and network services. The experts and vendors of these systems claim that this simplifies networking [7]. At its core, SDN offers higher flexibility and rapid routing of traffic flows. Within the framework of this separation, developers can utilize the control plane to change the behavior of the network without physical modification of the existing network infrastructure implementation. This allows developers to conduct experiments flexibly and efficiently and enables the rapid deployment of new network architectures. This architecture is visualized in Figure 1. Within the SDN architecture, the application layer provides users with a wide range of innovative services and applications, while the control layer is achieved by SDN software on the server. For ease of use, the SDN software includes a uniform application program interface (API) [8]. The data layer is comprised of generic network devices which are able to provide hardware or switching operations which are software defined in the control layer and communicated through the OpenFlow standard protocol [9].

The OpenFlow protocol is a fundamental element for building SDN solutions. It is the first standard communication protocol defined between the control layer and the infrastructure layer in SDN architecture [10, 11]. OpenFlow uses the concept of flows to identify network traffic based on matching rules that can be statically or dynamically programmed by the SDN control software. Switches are responsible for applying the proper actions on packets and updating records in the flow table entry. The switches simply forward packets according to the relevant entry in the flow tables without being concerned with how to construct or modify the flow table. The controller creates and installs a rule in the flow table for the corresponding packet if necessary, and the controller may at any time manage all switches by the flow table. OpenFlow-based SDN architectures provide extremely granular control, enabling the network to react to real-time changes from the application or the service user [12]. OpenFlow-based SDN technologies increase the bandwidth capability, dynamic nature of applications and significantly reduce operation and management complexity [13].

At present, the existing traffic scheduling algorithms mainly include Round-Robin scheduling algorithm and Greedy scheduling algorithm. These scheduling algorithms have some drawbacks, such as high cost, low reliability, and low scalability. The ability of data algorithms to deal with mass-traffic becomes more important with the increase in mobile user. To solve the complex selection problem that network faces, probability selection algorithm can be regarded as a kind of good method. For probability selection algorithm, the concerns are not a matter of a signal choice but the developing trend of server traffic and the load of servers. Briefly, in the part of the solution space, we get the existence of the optimal solution under complex environment. In each iteration, we save a set of candidate solutions and choose better feasible solution by using probability selection algorithm based on the mapping of server load and then produce a new generation of candidate solutions. The process is repeated until F-test value converges to the threshold.

3. Design and Implementation of Our Scheme

3.1. Load Balancing Technology. Load balancing provides a transparent way to increase the bandwidth of servers and other network devices and enhance data packet processing capacity and network throughput to ultimately improve the usability and flexibility of a network [14, 15]. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overloading any single resource. The importance of server load balancing is recognized such that methods to improve load balancing are actively and continually researched. In comparison to the rapid development of network technology, the growth rate of server processor speed and memory access is comparatively slow. At present, the processing overhead of servers is a

major bottleneck of the network development. Paradoxically, with the development of high-speed networks and increasing demands for services, many enterprise data centers and portal servers are becoming overwhelmed by the explosive growth in data traffic. Load balancing is the key technology used to distribute data demands across a cluster of server systems.

In this scheme of server load balancing based on a forward switching method, a novel method is proposed in this paper by utilizing the Network Address Translation (NAT) in the SDN architecture to construct a hybrid load balancing model. NAT refers to using a virtual address to represent the actual server address and rewriting the destination address of the request packet. Ultimately, data retransmission is performed [16, 17]. The present load balancing techniques are characterized by high investment, high consumption, low agility, and low reliability. Many of these issues can be solved by software-defined networking. This paper submits a new probability method of load balancing based on the variance analysis in SDN networks.

3.2. Variance Analysis

3.2.1. F-Test. Analysis of variance (ANOVA) is a set of statistical models which are used to analyze the differences between group means and their associated procedures, developed by R. A. Fisher. In the ANOVA setting, the observed variance of a particular variable is partitioned into components attributable to different sources of variation. This paper utilizes a variance analysis method to determine whether the averages of several sets of data are equal by analyzing data statistics.

For analyzing the statistical characteristics of port flux, this paper adopts the F-test method to detect whether there are significant differences among ports in order to determine if the operation is valid for the current state. Additionally, because data flow in the network is randomly selected, the traffic from each port can be viewed as independent with a normal distribution. The overall differences are divided into two basic classes of within-group variation and between-group variation. Differences in the between-group class are calculated to evaluate a meaningful dispersion between the average values of intragroup traffic and the population mean. Differences in the within-group class are calculated to evaluate the dispersion between an unbiased sample in the same group and the population mean. F-test analysis is a statistical technique that is used to identify a set of groups based on differences. The mean square is obtained through the calculation of differences between the two parts divided by their degrees of freedom. The F-inspection value is defined as the ratio of the "intra-" and "inter-" differences, according to the comparative analysis of the F-inspection and significance level threshold [18]. This will determine whether there is a significant difference between ports. Based on the above conclusions, the F-test formula is as follows:

$$F = \frac{\mathrm{MS}_b}{\mathrm{MS}_w} = F\left(\mathrm{df}_b, \mathrm{df}_w\right). \tag{1}$$

In (1), MS_b is the between-group difference, MS_w is the within-group difference.

To clarify by example, let $A_1, A_2, \ldots, A_k$ be a factor set having k different parts, let n_i be the number of monitoring times at level A_i, let X be related to the traffic, and let $X_{i1}, X_{i2}, \ldots, X_{in_i}$ be the set of X samples at level A_i. Consider

$$\begin{aligned} \overline{X} &= \frac{1}{N}\sum_{i=1}^{k}\sum_{j=1}^{n_i} X_{ij}, \quad i = 1, 2, 3, \ldots, k, \\ N &= \sum_{i=1}^{k} n_i. \end{aligned} \tag{2}$$

In (2), $\overline{X}$ equals the average of all of the traffic values. k refers to the number of groups. N refers to the number of the total monitoring times. Consider

$$\mathrm{SS}_t = \sum_{i=1}^{n}\sum_{j=1}^{k}\left(X_{ij} - \overline{X}\right)^2. \tag{3}$$

In (3), SS_t is the total sum of squares, which equals a square sum of deviations between every subsample in population and population mean. Consider

$$\mathrm{SS}_b = \sum_{j=1}^{k} n_j\left(\overline{X_j} - \overline{X}\right)^2. \tag{4}$$

In (4), SS_b is between-group sum of squares, which refers to a sum of squares of the deviations about the value between each group mean and population mean. Consider

$$\mathrm{SS}_w = \sum_{i=1}^{n}\sum_{j=1}^{k}\left(X_{ij} - \overline{X}\right)^2. \tag{5}$$

In (5), SS_w is within-group sum of squares, being equal to a sum of squares of the deviations about the value between every subsample value in group and each group mean. Consider

$$\begin{aligned} \mathrm{MS}_b &= \frac{\mathrm{SS}_b}{\mathrm{df}_b} \\ &= \frac{n_1\left(\overline{X_1} - \overline{X}\right)^2 + n_2\left(\overline{X_2} - \overline{X}\right)^2 + \cdots + n_k\left(\overline{X_k} - \overline{X}\right)^2}{N-1} \\ &= \frac{\sum_{j=1}^{k} n_j\left(\overline{X_j} - \overline{X}\right)^2}{N-1}, \end{aligned} \tag{6}$$

$$\begin{aligned} \mathrm{MS}_w &= \frac{\mathrm{SS}_w}{\mathrm{df}_w} \\ &= \left(n_1 - 1\right)S_1^2 + \left(n_2 - 1\right)S_2^2 + \cdots + \left(n_k - 1\right)S_k^2 \\ &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{k}\left(X_{ij} - \overline{X}\right)^2}{N-K}. \end{aligned} \tag{7}$$

In (6), the division of SS_b by the degree of freedom df_b returns a numeric result and assigns the result to MS_b which indicates within-group variance. Similarly, MS_w which refers

to between-group variance can be obtained according to the result of (7).

The F-test is employed to compare the factor of the total deviations. The F-inspection value is defined as the ratio of the "intra-" and "inter-" differences. An observed value of F which is greater than the critical value of F determined from tables indicates that there are significant differences among groups. Conversely, a small F-test value which does not exceed the critical value of F determined from tables indicates that there is no fundamental distinction among groups.

3.2.2. t-Test and Multiple Comparisons. Based on the results of the above calculations, we obtain the F-inspection value which can only be used to indicate whether there are significant differences among groups. The F-inspection value does not make it clear which of these groups, which should be few in number, contain noteworthy differences. There is a need to compare the calculated averages further by adopting the multiple t-tests. Before discussing the multiple t-tests, we first focus on two independent t-tests and assume that H_0: $\mu_0 = \mu_1$, H_1: $\mu_0 \neq \mu_1$. The t-test method expression is shown as follows:

$$t^2 = \frac{\left(\overline{X_0} - \overline{X_1}\right)^2}{\mathrm{SS}_w^2\left(1/n_0 + 1/n_1\right)} = \frac{n_0 + n_1}{n_0 n_1}\frac{\left(\overline{X_0} - \overline{X_1}\right)^2}{\mathrm{SS}_w^2}. \quad (8)$$

Then,

$$t = \sqrt{\frac{n_0 + n_1}{n_0 n_1}}\frac{\left|\overline{X_0} - \overline{X_1}\right|}{\mathrm{SS}_w}. \quad (9)$$

Let $n = n_0 + n_1$, and n denotes the sum of monitoring numbers.

According to the above principle, there is a formula of multiple t-test about k ($k > 2$) ports. As H_0 is true, the hypothesis is as follows:

H_0: $\mu_1 = \mu_2 = \cdots = \mu_k$, H_1: μ_i ($i = 1, \ldots, k$) are not all equal.

Then the multiple t-test method is with the formula as follows:

$$t_\alpha\left(\mathrm{df}_w\right) = \frac{\left|\overline{X_i} - \overline{X_j}\right|}{\sqrt{2\mathrm{MS}_w/n}}, \quad (10)$$

where $\overline{X_i}$, $\overline{X_j}$ are the point of any two of these averages; MS_w is the mean square and df_w is the degree of freedom. Consider

$$D_\alpha\left(\mathrm{df}\right) = t_\alpha\left(\mathrm{df}_w\right)\sqrt{\frac{2\mathrm{MS}_w}{n}}. \quad (11)$$

That is, if the difference between any two averages reaches or exceeds the significance level α, then the null hypothesis is rejected. It is then necessary to proceed effectively with dynamic load balancing to avoid contention.

3.2.3. The Existing Problems in t-Test. As to the comparison among the service port flux, when the number of groups is greater than 2, the probability of making type I error in a short period is increased.

When factor A consists of multiple independent ports, we can assume the following:

H_0: $\mu_1 = \mu_2 = \cdots = \mu_k$,

H_1: $\mu_1, \mu_2, \ldots, \mu_k$: not all μ_i are equal.

If H_0 is true, the counts of computation are $m = C_k^2 = k(k-1)/2$ times by using t-test. Now we suppose that the significance level is α; then the correct probability is $1 - \alpha$. In the meanwhile, through m series of comparisons, the probability of avoiding type I error is $1 - (1-\alpha)^m$. When the significance level is $\alpha = 0.05$ and the number of ports is $k = 4$, the probability of avoiding type I error is $p = 0.265$. When the significance level is $\alpha = 0.05$ and the number of ports is $k = 10$, the probability of avoiding type I error is $p = 0.402$. The error probability increases considerably.

3.2.4. The t-Test Adjustment of Multiple Comparisons. Research proves that analyzing the intragroup differences by using a significance level increases the probability of making type I error. Therefore we adjust the significance level; assume that the new significance level is 1 and then the following formula can be used:

$$\tilde{\alpha} = 1 - e^{(1/m)\ln(1-\alpha)}. \quad (12)$$

Substitute $\tilde{\alpha}$ into the following formula:

$$t_{\tilde{\alpha}}\left(\mathrm{df}_w\right) = \frac{\left|\overline{X_i} - \overline{X_j}\right|}{\sqrt{2\mathrm{MS}_w/n}},$$
$$D_{\tilde{\alpha}(\mathrm{df})} = t_{\tilde{\alpha}}\left(\mathrm{df}_w\right)\sqrt{\frac{2\mathrm{MS}_w}{n}}. \quad (13)$$

Finally, the minimum critical value is recalculated on a new significance level to ensure that the probability of making type I error can be controlled within a reasonable scope.

4. Selection Probability-Based Algorithm

Our ultimate aim is to reach a balance during transit from the source to its final destination and find another alternative server for releasing the overloaded one. The SDN controller modifies flow table entries for all possible switches in advance and sends flow tables to switches in time. By monitoring the flow direction, dynamic load balancing can be effectively implemented.

The stability of the network is analyzed with F-test. Lower F-test values indicate greater network stability. Hence network stability is inversely related to the F-test value which is adopted as a threshold parameter. The main process is illustrated as shown in Algorithm 1.

Step 1. If $F(\mathrm{df}_b, \mathrm{df}_w) > F(\alpha)$, calculate the minimum boundary value of differences $D_{\tilde{\alpha}(\mathrm{df})}$.

while $F(\mathrm{df}_b, \mathrm{df}_w) > F(a)$ do
  calculate the α' which replaces the primary α, set it to be our object parameter;
  find $t_{\alpha'}(\mathrm{df}_w)$ which is calculated by parameter α, make it as a variety to calculate the $D_{\alpha'}(\mathrm{df})$;
  while $D_{\alpha'(\mathrm{df})} > D(\alpha')$ do
    calculate probability parameters for all ports $\mathrm{Port}_i = \mathrm{Flow}(X_i)/\sum_{j=1}^{k} \mathrm{Flow}(X_j)$, and find Port_i among $(\mathrm{Port}_1, \mathrm{Port}_2, \ldots, \mathrm{Port}_i, \ldots, \mathrm{Port}_n)$, with the maximum port data flow;
    arrange the rest of the ports $\{P_1, P_2, \ldots, P_n\}$ in descending order. Generate a random number R and compare with the cumulative probability $q_i = \sum_{i=1}^{j} P_i$, and select the final variable j, j is the result as the index key;
  end while
end while

Algorithm 1

Table 1: Pairwise comparison among different ports.

	k_1	k_2	k_3	k_4
k_1		$**$	$*$	
k_2	$**$			$*$
k_3	$*$			
k_4		$*$		

Step 2. If $D_{\tilde{\alpha}(\mathrm{df})} > D(\alpha)$, populate ports ID which might have considerable differences in the comparison matrix (Table 1) integrated into our control module of SDN network.

The comparison matrix is shown in Table 1, where k_i $(1 \le i \le 4)$ denote horizontal ports. After the operation, the difference of the two means is compared with the minimum threshold. If the difference is greater than $\tilde{\alpha} = 0.05$, a symbol $*$ is used in the comparison matrix to indicate that there are differences between ports. If the difference is greater than $\tilde{\alpha} = 0.01$, the symbol $**$ is used to indicate that there are significant differences. The symbol $**$ is always a priority task for SDN controller.

Step 3. Perform operations with our algorithm based on a similar roulette wheel selection. Selection formula is as follows:

$$\mathrm{Port}_i = \frac{\mathrm{Flow}\left(X_i\right)}{\sum_{j=1}^{k} \mathrm{Flow}\left(X_j\right)}. \tag{14}$$

According to the traffic, a collection of port probability can be calculated: $\{\mathrm{Port}_1, \mathrm{Port}_2, \ldots, \mathrm{Port}_i, \ldots, \mathrm{Port}_n\}$.

Step 4. Arrange the probability in the descending order of the ranks.

Step 5. Compare a random number R that is uniformly distributed with cumulative probability. Thus, the obtained variable j represents the index of the selected port. Consider

$$q_j = \sum_{i=1}^{j} \mathrm{Port}_i. \tag{15}$$

Step 6. Repeat this first step. The real-time nature on SDN network is emphatically analyzed.

The important thing to note here is that performance overhead of control signals is not considered. We assume that there is no propagation delay.

For example, when the F-test value exceeds the significance level threshold, this indicates that the current network load is not balanced. Then we can find the port with the largest inflow traffic in the network by the t-test adjustment of multiple comparisons and can find the busiest server. At this time, the central controller in the network will perform traffic scheduling according to the algorithm based on selection probability, and the remaining traffic will be transferred to other servers.

It should be noted that the above probability selection algorithm is mainly for network traffic analysis and scheduling, and the overhead of SDN controller sending flow tables to SDN switches is not taken into account.

5. Module Implementation

5.1. ARP Processing. The Address Resolution Protocol (ARP) is a telecommunication protocol used for resolution of network layer addresses into link layer addresses, which is a critical function in multiple-access networks. In the SDN network, on the client side, before sending an HTTP request, first send a gratuitous ARP to the Openswitch [19]. The Openswitch does not get a matching table and generates a Packet-In message sent to the SDN controller. The load balancing module, which is integrated into the controller, will resolve the Packet-In message. A new ARP packet which is filled in the destination address, IP and forwarding port information, and so on will be reassembled in Packet-out forms and sent to Openswitch again. The client receives a new ARP reply packet and accepts the ARP entry into its ARP table. The ARP request processing is carried out by our load balancing module.

5.2. TCP Request. For end-users accessing the site, load balancing makes all servers appear as a single server with a single IP address; all load balancing is transparent. When the Openswitch receives the initial HTTP access request, it does not have a matching table and generates a Packet_In message which is sent to the SDN controller [20]. The controller parses the message and reassembles the flow table as shown in

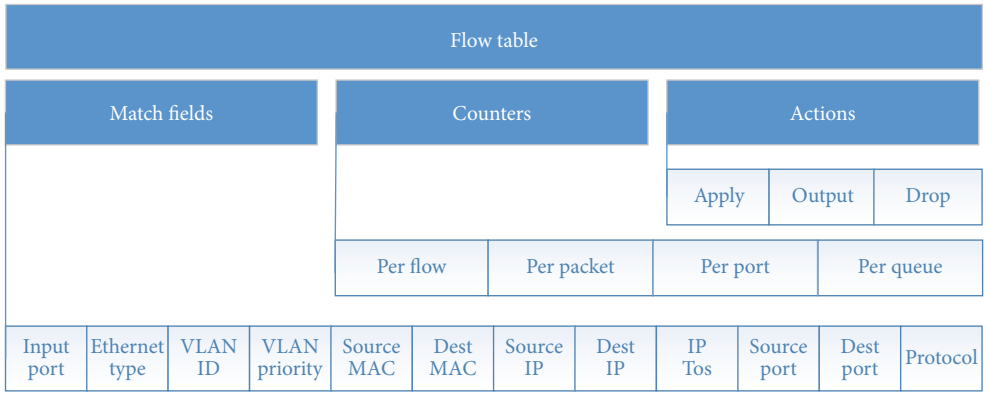

Figure 2: Package format of the flow table.

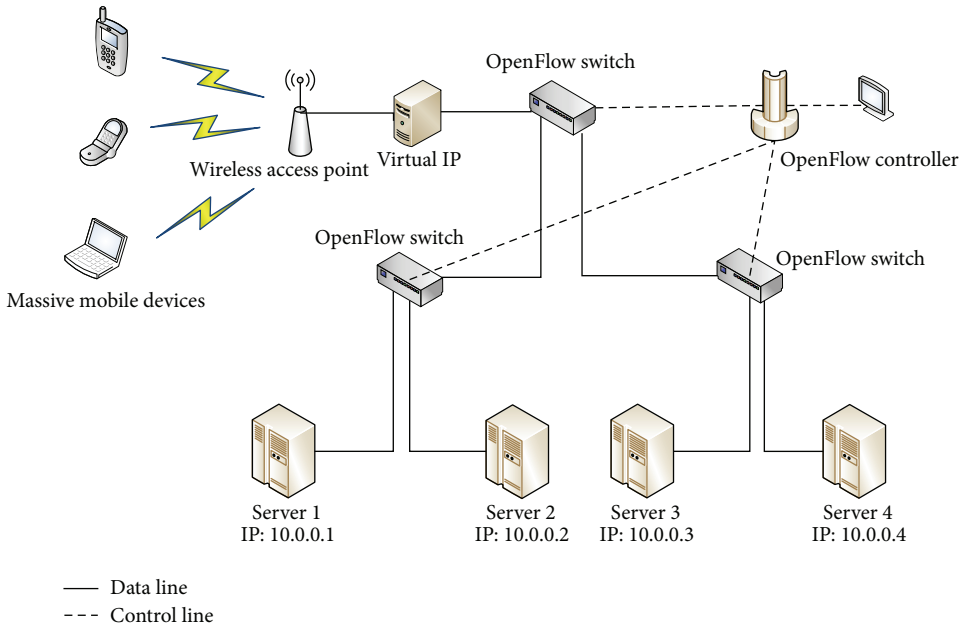

Figure 3: Test platform set up to measure model realization.

Figure 2 using object methods in the SDN controller such as OFMatch, OFAction, and OFFlowMod. On deployment, the flow table is sent to the Openswitch. This process will replace the virtual address with the physical address.

6. Experiment Result and Performance Analysis

In the experiment, our operating system was Ubuntu 14.04.3-desktop-amd64, controller was Floodlight version 1.0, Mininet 2.1.0 which is a network emulator for the creation of virtual network using the Ubuntu kernel was used to define the topology of the whole network, and Open vSwitch 2.3.2 was used to simulate the required OpenFlow switch. In order to measure the experiment, an OpenFlow test platform was built as depicted in Figure 3. The SDN network model contains four independent server nodes, three OpenFlow switches, and a Floodlight controller. Our experiment presented some server code written in Python, but almost the same design would apply for nearly any language. Python came with a simple platform built in HTTP server.

In the experiment, Mininet was to first build architecture with different paths. Each server represented a physical machine in Mininet and had its own actual IP address. We created a virtual IP address which is advertised from the NAT, and incoming traffic destined to this virtual IP address was routed by Floodlight controller to different actual IP addresses. We created automated scripts of access requests on the clients. Next, we supplied real-time traffic flux statistical analysis through the analysis of variance and traffic scheduling algorithm modules which were integrated into the Floodlight controller and chose appropriate path based on our analysis result.

These switches do not limit the transmission speed and work with maximum link rate. Java code was used to implement the analysis of variance and traffic scheduling algorithm modules which are integrated into the Floodlight controller.

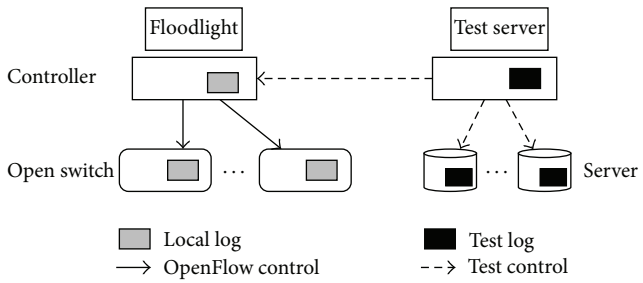

Figure 4: Test platform structure.

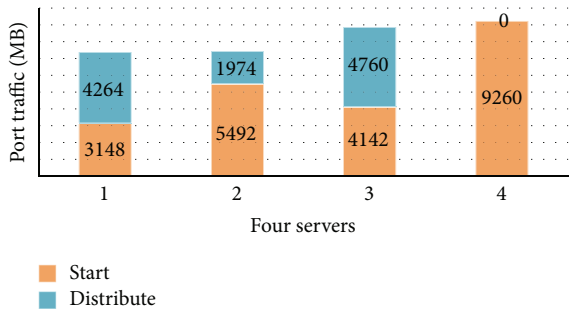

Figure 5: Probability scheduling algorithm.

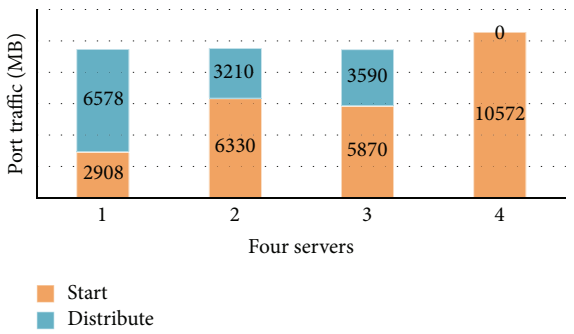

Figure 6: Greedy scheduling algorithm.

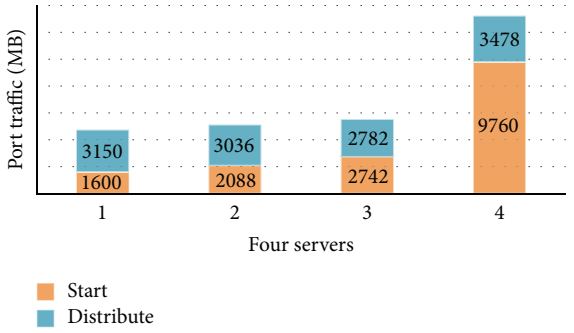

Figure 7: Round-Robin scheduling algorithm.

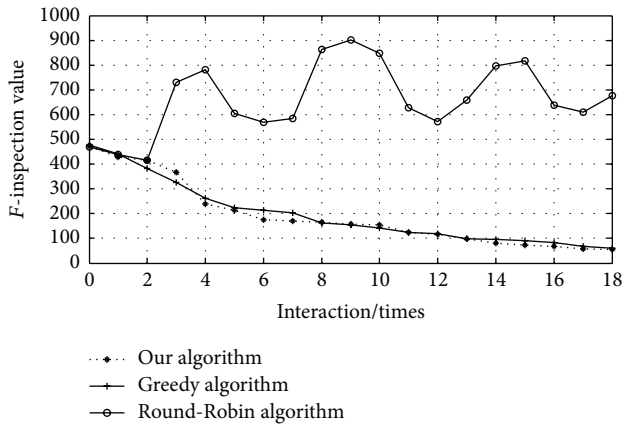

Figure 8: The F-test values of three scheduling algorithms.

By analyzing the results of the experiment and comparing with other algorithms, the improvement in the performance of the proposed algorithm was verified. The experiment consists of initialization of data flow, traffic analysis of variance, and calling of the load balancing algorithm.

The experiment requires a measurement of quantitative analysis based on a measurement server in the SDN. The controller can save data plane information and interplay relationship between controller and OpenFlow switch to a local log file. The measurement server (testing server) executes synchronization operation on the Floodlight controller and servers and gets the traffic data. The general arrangement of the testing platform is shown in Figure 4.

Through a simulation experiment, the performance of the proposed algorithm is verified. Assume that a user requests access to a virtual address. The communication time of each server including a web service request is 5 seconds.

When excessive server load occurs on one server, the following three load balancing algorithms are executed individually: Round-Robin scheduling, Greedy scheduling, and probability scheduling. The traffic at each server is captured by the variance analysis module in the Floodlight controller and this piece of information is saved to log files. Figures 5, 6, and 7 illustrate the starting and ending positions of port traffic. Figures 5 and 6 show the response of the proposed algorithm and the Greedy algorithm, respectively. It can be seen from these figures that the peak values of all the columns are essentially flat, meaning that all four servers are able to load balance using by the two methods. In a real environment, it is not necessary to keep strict equilibrium at any time. Figure 7 shows the result of the Round-Robin scheduling algorithm. There is considerable difference in the peak values of each column, which indicates that this method is ineffective.

Figure 8 displays curves for all scheduling algorithms in one figure and their relationship with the F-test value. The smaller the values of the F-test in a series of experiments are, the fewer the differences among the monitoring data traffic of each port are. As shown here, the proposed algorithm matches well with the Greedy algorithm. With the increase of the times of interaction transmission, the F-test values are evidently reduced, and these methods can modify all servers load balancing exactly in real-time and redirect traffic more efficiently. It is also clear from the figure that the Round-Robin algorithm is not the best one which means there are

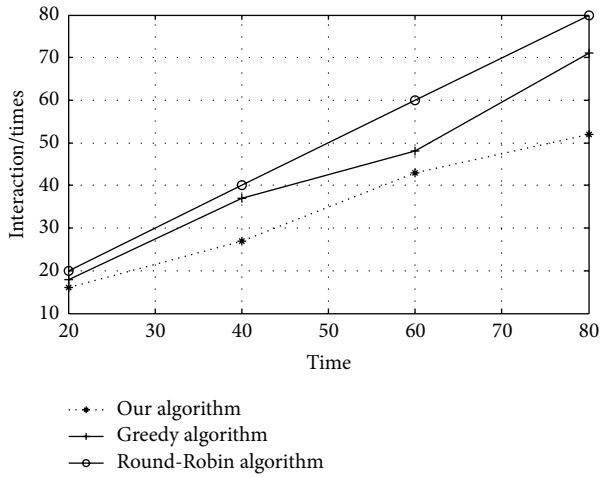

Figure 9: Interaction times of three scheduling algorithms.

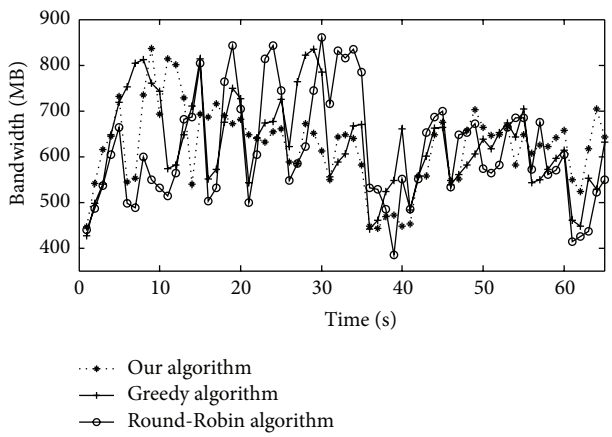

Figure 10: Link bandwidths of three scheduling algorithms.

great differences among servers' load, and this method is unbalanced.

Figure 9 demonstrates the interaction of controller and switch based on the above three methods. The simulation result shows that the proposed algorithm requires less interaction information than the other two because the selection algorithm is based on a probability model. There is always a probability that the controller will calculate the same flow table as implemented previously and the controller does not need to interact with switches when this occurs.

Figure 10 shows link bandwidth of the each server node while applying the three scheduling algorithms. The proposed algorithm in this paper shows that the bandwidth line remains flat or increases at 11 s, 16 s, and 41 s. The reason for this is that the proposed algorithm is based on probability selection, in which the state does not always change but is adjusted by probability selection. In the next moment there is a certain probability to reconnect the previous link. Then, the direct impact on link bandwidth should not be significant. These cases have been shown at 11 s, 16 s, and 41 s in Figure 10.

But we can see that the link bandwidth of the Round-Robin scheduling algorithm decreases after every 5-second request. Every time the controller will redirect traffic to another server and cause the decline of instant rotational speed. The Greedy algorithm appears to be increasing at 6-second intervals. The reason is that the currently selected port is just the previous selection, without needing to choose a port. So the bandwidth is not turning out. Nonetheless after that the curve does not appear with the next bandwidth higher than before selection. This is because the flow table is recreated every time after a minimum selection.

The same requests to three scheduling algorithms return different results at each invocation. Each oscillation curve represents different bandwidth fluctuation. The experimental figure demonstrates that the bandwidth fluctuation using the Round-Robin and the Greedy algorithm is with more volatility than the proposed algorithm. We assume variable R which is a fluctuation coefficient. This variable can be obtained by using following formula:

$$R = \sum \frac{|B(t) - B(t-1)|}{n}, \quad t = 5 * i \ \ (i = 1, 3, \ldots, 2n-1), \ n \in N. \tag{16}$$

The variable R is smaller and the bandwidth fluctuation is steadier (by calculating, our algorithm can reduce 18.2% volatility rate of the Greedy algorithm and 34.4% volatility rate of the Round-Robin algorithm).

7. Conclusion

In this paper, a SDN load balancing scheme based on variance analysis is proposed for supporting dynamic workloads from massive mobile users by using the advantage of the separation of SDN control layer and data layer. We have descried the concrete algorithm and every part of the load balancing module. Meanwhile, we test the method and give the result of comparison in SDN environment. Based on the OpenFlow technology, we present the method of variance analysis to monitor data traffic of each server port and dynamically redirect traffic dynamically with the OpenFlow technology. The dynamic demand from massive mobile users could be satisfied at run-time by using the probability-based selection algorithm which is integrated with the SDN controller. Our method can efficiently monitor dynamic workload from mobile networks and achieve better load balancing when compared to the existing solutions, so as to achieve good scalability to support massive mobile users. SDN simplifies the network; it enables traffic to be intelligently controlled for quick reaction to dynamic network conditions. In the future, we will focus on the "mobile SDN" and enhancing load balancing efficiency to be delivered through highly automated orchestration, new technologies such as network virtualization, lifecycle service orchestration.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The work was supported by the National Natural Science Foundation of China (no. 61173188, no. 61572001, and no. 61502008), the Research Fund for the Doctoral Program of Higher Education (no. 20133401110004), the Educational Commission of Anhui Province, China (no. KJ2013A017), the Natural Science Foundation of Anhui Province (no. 1508085QF132), the Tender Project of the Co-Innovation Center for Information Supply & Assurance Technology of Anhui University (no. ADXXBZ2014-7), and the Doctoral Research Startup Funds Project of Anhui University.

References

[1] Y. Wu, G. Min, and L. T. Yang, "Performance analysis of hybrid wireless networks under bursty and correlated traffic," *IEEE Transactions on Vehicular Technology*, vol. 62, no. 1, pp. 449–454, 2013.

[2] D. Zhang, H. Huang, J. Zhou, F. Xia, and Z. Chen, "Detecting hot road mobility of vehicular Ad Hoc networks," *Mobile Networks and Applications*, vol. 18, no. 6, pp. 803–813, 2013.

[3] M. Dong, T. Kimata, K. Sugiura, and K. Zettsu, "Quality-of-Experience (QoE) in emerging mobile social networks," *IEICE Transactions on Information and Systems*, vol. 97, no. 10, pp. 2606–2612, 2014.

[4] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[5] S. Sezer, S. Scott-Hayward, P. Chouhan et al., "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[6] M. Dong, H. Li, K. Ota, and J. Xiao, "Rule caching in SDN-enabled mobile access networks," *IEEE Network*, vol. 29, no. 4, pp. 40–45, 2015.

[7] M. Channegowda, R. Nejabati, and D. Simeonidou, "Software-defined optical networks technology and infrastructure: enabling software-defined optical network operations [invited]," *Journal of Optical Communications and Networking*, vol. 5, no. 10, Article ID 6645122, pp. A274–A282, 2013.

[8] M. K. Shin, K. H. Nam, and H. J. Kim, "Software-defined networking (SDN): a reference architecture and open APIs," in *Proceedings of the International Conference on ICT Convergence (ICTC '12)*, pp. 360–361, IEEE, October 2012.

[9] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[10] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[11] S. Huang, J. Griffioen, and K. L. Calvert, "Network hypervisors: enhancing SDN infrastructure," *Computer Communications*, vol. 46, pp. 87–96, 2014.

[12] M. Kobayashi, S. Seetharaman, G. Parulkar et al., "Maturing of OpenFlow and software-defined networking through deployments," *Computer Networks*, vol. 61, pp. 151–175, 2014.

[13] H. Yin, T. Zou, and H. Xie, "Defining Data Flow Paths in Software-Defined Networks with Application-Layer Traffic Optimization," US Patent Application 13/915,410[P], 2013.

[14] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '10)*, pp. 551–556, IEEE, Perth, Australia, April 2010.

[15] G. Min, Y. Wu, and A. Y. Al-Dubai, "Performance modelling and analysis of cognitive mesh networks," *IEEE Transactions on Communications*, vol. 60, no. 6, pp. 1474–1478, 2012.

[16] C.-C. Yeh, C.-W. Huang, and T.-H. Lin, "A new network address translation traversal mechanism design and implementation," *Advanced Science Letters*, vol. 20, no. 2, pp. 496–500, 2014.

[17] Y. Wu, G. Min, and A. Y. Al-Dubai, "A new analytical model for multi-hop cognitive radio networks," *IEEE Transactions on Wireless Communications*, vol. 11, no. 5, pp. 1643–1648, 2012.

[18] B. Feng and X. Zhang, "Study on t-test, analysis of variance and multiple comparisons," *Journal of Taiyuan Normal University (Natural Science Edition)*, vol. 11, no. 4, pp. 46–49, 2012.

[19] Q.-Y. Zuo, M. Chen, G.-S. Zhao, C.-Y. Xing, G.-M. Zhang, and P.-C. Jiang, "Research on OpenFlow-based SDN technologies," *Journal of Software*, vol. 24, no. 5, pp. 1078–1097, 2013.

[20] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou, "OrchSec: an orchestrator-based architecture for enhancing network-security using network monitoring and SDN control functions," in *Proceedings of Network Operations and Management Symposium (NOMS '14)*, pp. 1–9, IEEE, Krakow, Poland, May 2014.